



Procedia Computer Science

Volume 29, 2014, Pages 1468–1479

ICCS 2014. 14th International Conference on Computational Science



High Performance Message-Passing InfiniBand Communication Device for Java HPC

Omar Khan, Mohsan Jameel and Aamir Shafi

SEECs, National University of Sciences and Technology, Islamabad, Pakistan

{11mscsokhan, mohsan.jameel, aamir.shafi}@seecs.edu.pk

Abstract

MPJ Express is a Java messaging system that implements an MPI-like interface. It is used for writing parallel Java applications on High Performance Computing (HPC) hardware including commodity clusters. The software is capable of executing in multicore and cluster mode. In the cluster mode, it currently supports Ethernet and Myrinet based interconnects and provide specialized communication devices for these networks. One recent trend in distributed memory parallel hardware is the emergence of InfiniBand interconnect, which is a high-performance proprietary network and provides low latency and high bandwidth for parallel MPI applications. Currently there is no direct support available in Java (and hence MPJ Express) to exploit the performance benefits of InfiniBand networks. The only option to run distributed Java programs over InfiniBand networks is to rely on TCP/IP emulation layers like IP over InfiniBand (IPoIB) and Sockets Direct Protocol (SDP), which provide poor communication performance. To tackle this issue in the context of MPJ Express, this paper presents a low-level communication device called *ibdev* that can be used to execute parallel Java applications on InfiniBand clusters. MPJ Express is based on a layered architecture and hence users can opt to use *ibdev* at runtime on an InfiniBand equipped commodity cluster. *ibdev* improves Java application performance with access to InfiniBand hardware using native verbs API. Our performance evaluation reveals that MPJ Express achieves much better latency and bandwidth using this new device, compared to IPoIB and SDP. Improvement in communication performance is also evident in NAS parallel benchmark results where *ibdev* helps MPJ Express achieve better scalability and speedups as compared to IPoIB and SDP. The results show that it is possible to reduce the performance gap between Java and native languages with efficient support for low level communication libraries.

Keywords: MPJ Express, Java MPI, Java InfiniBand

1 Introduction

Java is a popular programming language because of its portability, security, and wide-spread knowledge. It is beginning to play a role that C and C++ have been performing successfully for many

years, especially in enterprise computing applications. Java is fast turning into a system level programming language instead of just an internet specific technology [1].

Initially, Java was heavily censured for its poor execution speed compared to native languages like C, C++ and Fortran [2]. But, in the past many years, a lot of effort has been made to enhance the performance of Java applications. The most successful approach in this context is the Just-in-Time (JIT) compiler within the JVM. It caches the most frequently used native code and remarkably cuts down byte-code to native code translations, in turn increasing code execution tremendously. Another reason for the performance increase of Java is the close collaboration between microprocessor and JVM designers. Since 2007, Oracle and Intel have worked together and improved Java's performance by up to 14 times by correlating language optimization with hardware design [8]. Intel has benchmarked a TeraSort application on an Intel Xeon powered Hadoop cluster and showed a performance gain of 50 percent with JDK 7 update 13 compared to JDK 6 update 14 [8].

In recent years, more and more Java server applications have been deployed in data centers. One such success story is the Apache Hadoop software, which is a scalable, distributed data storage and analytics engine and can process huge amounts of data. The core components in Hadoop: Apache Hadoop Distributed File System, MapReduce, Apache Hadoop common are all written in Java. Currently many leading organizations including Facebook, Yahoo, Amazon, and IBM are relying on Hadoop software to run their business applications.

Apart from increasing instruction execution speed of software, a lot of research is also under way to develop high speed proprietary communication interconnects. These interconnects deliver low latency and high bandwidth to user applications. InfiniBand is one such interconnect for network and inter-process communication. The number of InfiniBand-based supercomputers has increased to 207, signifying 41.4 percent of the TOP500 clusters. According to the list issued in June 2013, InfiniBand is used to connect 16 out of 33 peta-scale capable systems.

If Java is to become a language of choice for HPC community, the tremendous improvement in its computational performance needs to be augmented with high communication performance on these new network architectures. Presently, no built-in support is available in Java to take benefits of the communication capabilities of InfiniBand interconnect. Java applications depend on Upper Layer Protocols (ULPs), IPOIB and SDP, to run on InfiniBand network.

However, IPOIB and SDP have a high overhead and degrade communication performance of the application significantly. According to Zhang [7], in a Gigabit Ethernet environment, the latency and bandwidth of Java bench-marks is very close to C counterparts, though with a higher CPU utilization. However, since Java can only run on InfiniBand using ULPs, the test results of Java benchmarks over InfiniBand network show big gaps in bandwidth and latency when compared with C counterparts.

Message-passing Interface (MPI) is considered the *de facto* standard for building parallel applications on HPC hardware [10]. Performance and scalability of an MPI application is heavily dependent on the rate at which messages can be exchanged between processes running on different cluster nodes. The programming languages for development of MPI applications have predominantly been C, C++ and Fortran. But, Java also offers an attractive alternative, as it has support for networking, multi-threading and built-in security features. In this context, MPJ Express is a popular MPI-like library that currently implements the mpiJava 1.2 API [16]. The software is capable of executing in two modes. The first mode called the cluster mode is used on commodity clusters built using a fast interconnect. The second mode called the multicore mode is used when MPJ Express is used for running parallel applications on shared memory parallel systems like Symmetric Multi-Processors (SMPs) or multicore processors. In the context of cluster mode, the current release of MPJ Express provides two transport level devices. There is a Java NIO based device called *niodev* that is geared towards Ethernet networks and Myrinet eXpress (MX) based device called *mxdev* for Myrinet interconnect.

Java message passing libraries have predominantly provided portable communication devices, rather than focus on developing low level devices for communication on high speed networks. This

has resulted in a lower performance compared to native MPI libraries. To achieve optimal performance in a cluster, it is very important to implement messaging efficiently for the particular proprietary interconnects like InfiniBand.

Until now, MPJ Express can be run over an IB network via IPoIB or SDP. Performance tests have been run on both IPoIB and SDP. When MPJ Express is run on a 40 Gbps InfiniBand network in cluster configuration (*niodev*), SDP gives a maximum bandwidth of 7.4 Gbps while IPoIB gives a nominal 1.1 Gbps. This is primarily due to extra overhead of ULP's. These results highlight the significance of a new low level device, we call *ibdev*, which can make full use of IB network and give Java network applications a reasonable speedup. *ibdev* relies on the low level Verbs API instead of ULP's. This allows MPJ Express direct access to the InfiniBand messaging service which results in much improved communication performance.

Our performance evaluation reveals that MPJ Express achieves latency of 27 micro-seconds using *ibdev* compared to 51 micro-seconds using IPoIB—for one Byte message. On the other hand, MPJ Express achieved throughput of 16 Gbps using *ibdev* as compared to 7.4 Gbps using SDP—for 2 Mega Byte message. Improvement in communication performance of MPJ Express is also depicted in NAS parallel benchmark (NPB) results. NPB are one of the most well-known benchmarks for evaluating performance of high performance parallel systems. Here too, *ibdev* helps MPJ Express achieve better scalability and speedups compared to the ULPs.

The structure of the paper is as follows: Section 2 gives an overview of some related work. Section 3 introduces a high level design of MPJ Express. Section 4 explains the architecture and implementation details of our communication device, *ibdev*. Section 5 discusses the performance results on InfiniBand cluster. The evaluation includes point-to-point micro-benchmarking as well as NAS parallel benchmarks. Section 7 gives a conclusion and discusses future work.

2 Related Work

This section discusses the most recent approaches that have been adopted to enhance performance of Java applications running on IB interconnect.

FastMPJ [5] is a message-passing communication library that supports shared memory and high-speed networks. FastMPJ includes communication devices on top of MX/Open-MX, InfiniPath PSM and IB Verbs native libraries. It obtains performance similar to MPI libraries because of efficient and scalable communication protocol implementations, topology aware collective primitives, zero-copy protocols to avoid buffering overheads, and efficient shared memory communication using threads.

mpiJava is an object oriented interface to MPI in Java. It is implemented as a collection of JNI wrappers to native MPI libraries. mpiJava had been inactive for the past few years but OpenMPI has recently taken it up and integrated it in their latest developer trunk of the library. However, these new Java bindings would only work with OpenMPI, as it would require a huge effort to wrap a number of functions for various MPI libraries.

MPJ/Ibis [12] implements the Java Grande Forum (JGF) MPJ API over the Ibis grid programming platform. Ibis's flexibility allows to run java message-passing applications on clusters without recompiling. All required modules are loaded by the respective communication platform library at runtime. This library can use either “pure” Java communications, based on Java sockets, or native communications on Myrinet. However, MPJ/Ibis is not thread-safe and its Myrinet support is based on the GM library which has a lower performance than MX library. It also lacks direct InfiniBand support and relies on ULP's to run on InfiniBand.

Jdib [14] is a Java wrapping of InfiniBand Verbs API. It provides high performance by directly using the InfiniBand RDMA mechanism through Java Native Interface (JNI). Jdib provided Java developers with an RDMA API. It has a much better performance than the solutions based on IPoIB and SDP. The main shortcoming of Jdib is its low level API which incurs a JNI overhead for each call.

For each structure declared in verbs API, Jdib creates a matching Java class whose members map to the members of the equivalent C structure.

FastMPJ is currently the most stable and successful Java Message-Passing library. It is a very efficient and high performance implementation of mpiJava 1.2 on InfiniBand networks, but it is not open-source and one needs to purchase a license to use it. Nevertheless, FastMPJ highlights the benefits of implementing support for low level communication libraries in Java. Jdib also uses IB verbs library but the API is too low level and relies heavily on JNI for each operation to be effectively used in a parallel application.

3 MPJ Express Design and Architecture

MPJ Express has a layered architecture that provides the capability to add/update and swap layers in or out as needed, as long as the interfaces are not altered. Figure 1 is a view of various layers in the MPJ Express software. This section is an overview of the responsibilities of the different layers within MPJ Express. It highlights the abstractions that each layer provides to the one lying above it.

A Java application uses point-to-point and collective routines of MPJ Express for message-passing. These routines in turn call *mpjdev* layer which runs over the *xdev* layer. The *mpjdev* layer is a middleware between the high level communication API and the low level *xdev* layer; which implements the actual network device. *xdev* layer also has a pluggable architecture that provides a simple but comprehensive communication API. This design facilitates the development of new communication devices using platform optimized native libraries.

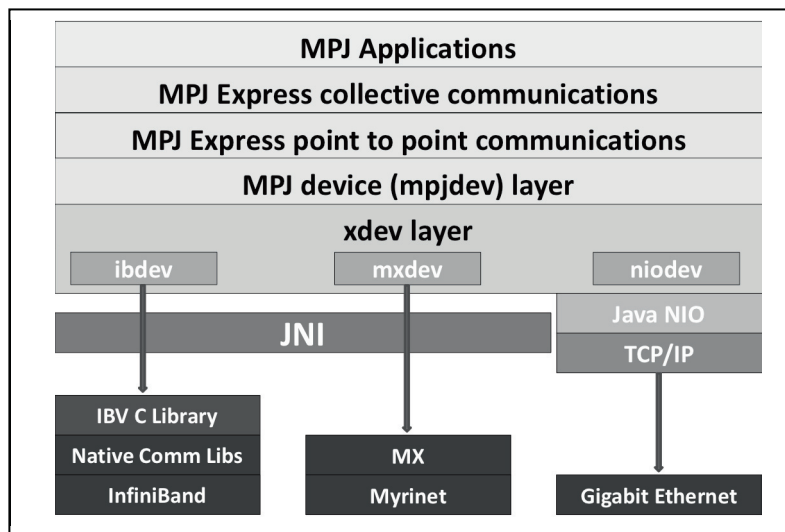


Figure 1: MPJ Express design including InfiniBand device

The *xdev* layer of Figure 1 forms the MPJ Express Infrastructure. This is based on the notion of device drivers, similar to that of MPICH. MPJ Express uses various different communication devices on top of which higher-level operations like point-to-point and collective communications are implemented. This approach allows different device drivers for different interconnects, For example, Java NIO, Java I/O or Myrinet Express. The *xdev* API includes point-to-point communication routines only. It has both blocking and non-blocking communication primitives along with synchronous mode.

The *xdev* API is listed in Table 1 below. As can be seen, *xdev* API has only a few basic point-to-point primitives on which are based complex collective communication primitives of the higher layers.

The two layers in Figure 1, below the MPJ application, form the messaging API. It provides an interface to the application developers to write message-passing parallel applications. This interface hides all the underlying implementation details. The messaging API consists of the point-to-point communications, collective communications, and some utilities like communicators, process topologies, and derived data-types.

```
public abstract class Device {
    //Initialization
    ProcessID[] init(String[] args);
    ProcessID id();

    //Blocking communication primitives
    void send(Buffer buf, PID dest, int tag, int context);
    Status recv(Buffer buf, PID src, int tag, int context);
    void ssend(Buffer buf, PID dest, int tag, int context);

    //Non-Blocking communication primitives
    Request isend(Buffer buf, PID dest, int tag, int context);
    Request issend(Buffer buf, PID dest, int tag, int context);
    Request irecv(Buffer buf, PID src, int tag, int context, Status s);
    Status probe(PID src, int tag, int context);
    Status iprobe(PID src, int tag, int context);
    Request peek();

    //Clean up resources
    void finish();
}
```

Table 1: *xdev* API

4 InfiniBand Communication Device for MPJ Express

This section gives high level design of the new communication device, *ibdev*. An overview of the communication model of InfiniBand is also presented.

MPJ Express communication infrastructure (*xdev*) also has a layered architecture. The first of these layers is the definition of the new device, *ibdev*. This is a pure Java class. *ibdev* uses JNI to access native code. The next layer in the infrastructure is the IBV library in C. IBV library provides *ibdev* with a set of communication functions based on low level InfiniBand specific native libraries. This layered design greatly eases the development of *xdev* communication device. Current release of MPJ Express also has support for Myrinet interconnect via *mxdev*, which also uses JNI to access MX library. However, InfiniBand does not have a similar library that may provide the functions that need to be implemented in *xdev*. The last layer in the communication middleware is *libibverbs*, which is an implementation of the IB verbs API and *librdmacm*, which is an RDMA connection library. *libibverbs* handles creating, modifying, querying and destroying of resources.

The JNI layer of *ibdev* acts as a wrapper over IBV C library. The JNI code has been kept to a minimum to avoid extra overhead. All native methods of *ibdev* call native IBV C library functions

through JNI, implementing a series of steps: (a) get Java objects necessary for calling the native function; (b) call native library function; and (c) save results in the corresponding Java objects. In order to minimize the overhead associated with JNI calls, caching of object references has been extensively used.

4.1 Channel and Memory Semantics

InfiniBand has support for both channel semantics and memory semantics. In channel semantics communication style, one process sends the data and the receiving process decides the destination of the incoming data. The sender only knows about the destination's Queue Pair and not the memory address where the sent data will be written.

With memory semantics, also called Remote Direct Memory Access (RDMA), the sender directly reads or writes the virtual memory of a remote process. The receiver only needs to inform the sender of the memory address; it is not involved in the data transfer.

The C performance benchmarks that come with IB Verbs, show that bandwidth for large messages (> 32K) of both send/receive and RDMA operations is the same. However, send/receive have a higher latency (small messages) and also consume more CPU time, therefore, *ibdev* has been implemented with RDMA semantics.

4.2 Implementation of IBV C Library (Native Code)

This section describes high level implementation details of the native C library in *ibdev*. It highlights some of the core features of the library. The library functions have a one to one mapping with the *xdev* functions in Java.

IBV library is a C library built on top of the IB Verbs API. It implements a set of low-level communication primitives, which are in turn used by the higher layers. The type of transport service used to set up the Queue Pairs (QP) for communication between processes is very important. A QP provides the transport layer with a specific transport service. Different transport services have various reliability levels for connected and connectionless communication. The various transport services provided by InfiniBand are: Reliable Connection, Unreliable Connection, Reliable Datagram, Unreliable Datagram and RAW Datagram. In case of IBV library, Reliable Connected transport service has been selected because it guarantees that messages are acknowledged, delivered in order, without corruption and at most once.

Initialization: In order for processes in the system to communicate with one another, an all-to-all connection needs to be established between them. The Connection Manager library (`librdmacm`) manages the protocols and mechanisms required to establish, maintain, and release channels for the IB Reliable Connection. CMs maintain a certain amount of information for the lifetime of a connection. *xdev* does not work with high level MPI ranks and groups. At the device level there are unique Process ID's. These are inherited from Java UUID class. At connection setup, these Process ID's are exchanged between all processes and associated with a Queue Pair Number (QPN). These Process ID's greatly ease the implementation of the *xdev* layer, as this layer is no longer concerned with communicators, ranks, and groups existing at the MPI level.

Buffering: The IB Architecture requires pinning of buffers prior to communication. This involves registration of the buffer with hardware. Buffer registration is an expensive operation and involves both OS kernel and the host's NIC. This operation needs to be avoided for eager protocol. This problem is addressed using a pool of pre-registered buffers. When a message is to be sent, it is copied into the already registered buffer. MPJ Express has a configurable protocol shift length. It is the buffer size after which the device switches from eager to rendezvous protocol. At startup a process allocates several buffers and registers them with the OS. The remote keys of these buffers are also exchanged between processes via control messages. After this, any time processes wants to send data less than the protocol shift length, this pre-registered buffer pool is used. One of the issues with

RDMA write is detection of incoming RDMA write at the receiver. For RDMA write operations, no completion notification is generated at the receiver. The mechanism of detecting incoming data is to continuously poll the contents of the destination buffer. In *ibdev* the receiver side writes a 64 bit random number near the end of the receive data buffer whenever it is done reading data from it. When receive method is called the process continuously polls this buffer location. When sender writes its data to this memory address, and the random number is overwritten: this indicates completion of write operation. These random numbers are created at connection setup and assigned for each process.

Control messages are exchanged to signal start and end of communication. These messages keep the sender and receiver in knowledge of each other's current status. The buffer information along with the remote memory keys are also exchanged via these control messages. These memory keys allow a process to remotely write to the memory of another process. These messages are exchanged via send/receive semantics of IB Verbs. The small size of a control message allows each process to post a number of receive work requests in the receive queue, so the possibility of an empty receive work queue is non-existing. Each process also reposts receive request for a control message soon as one is consumed. Unlike MPI 2.0 specifications, which require that buffers are pre-pinned for RDMA operations, mpiJava 1.2 API specifications do not have the buffer pinning option. For large static buffers or dynamic buffers, the approach of pre-allocating and pre-pinning buffers, as in eager protocol, is not possible. In this case, the buffers are pinned on the fly. For any send and receive operation, buffers need to be registered before carrying out the operation. The memory keys of these registered buffers are communicated to the remote process via control messages.

Non-Blocking calls return immediately to the calling function. The actual data transfer is done in the background. User can call any number of non-blocking sends/receives. In *ibdev* non-blocking calls to different processes are carried out in parallel and completions are reported independently. However, non-blocking calls to same process are carried out sequentially. We need a mechanism to process non-blocking calls efficiently. The easiest method for implementing non-blocking communication using multiple threads is to spawn a new thread for each send/receive request. The caller can then return immediately. However, creation/termination of threads is an expensive operation. The solution, implemented in *ibdev*, is to create a thread pool. It consists of a fixed number of threads and delegates incoming jobs to these threads in an efficient way. The number of threads used is a parameter that can be tuned to provide the best performance.

5 Performance Evaluation

This section presents a comparison of the performance of *ibdev*, with existing MPJ Express network communication device, *niodev*, over InfiniBand. The comparison consists of benchmarking point-to-point data transfers as well as an analysis of the performance of NAS parallel benchmarks.

The experimental environment used for performance testing comprises of 32 node cluster hosted at supercomputing lab at RCMS, National University of Sciences and Technology. Each node has two Intel quad-core Xeon E5520 processors with 24 GBytes of main memory. All nodes are connected by 40 Gbps QDR InfiniBand Interconnect for internal communication. *ibdev* is integrated into MPJ Express version 0.38. Oracle JDK 1.7.0_25 was used for Java and GNU GCC 4.8.1 for C compilation was used. OpenMPI version 1.9a1r29075 was used for C MPI implementation.

5.1 Point-to-point MPJ Express performance

Figure 2 and 3 shows the latency and throughput comparison across InfiniBand. The latency of C MPI library (OpenMPI) is the lowest at 6 microseconds for 1 Byte message size. MPJ Express running *ibdev* has 28 microseconds. MPJ Express running over *niodev* using ULPs i.e. IPoIB and SDP has much higher latencies at 80 microseconds and 300 microseconds respectively. The difference between

C MPI library and MPJ Express running on *ibdev* is due to overhead incurred by internal buffer layer as it can be seen that when MPJ Express is run over *ibdev* without buffering layer the latency stands at 8 microseconds closer to the C MPI implementation. The thin JNI layer does not contribute towards the performance degradation from C implementation.

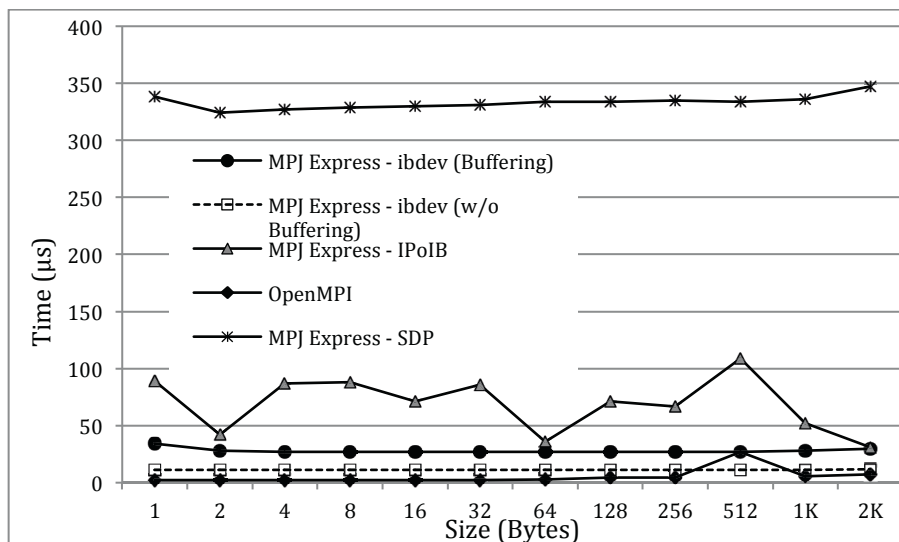


Figure 2: Latency of Message Passing Libraries

The throughput comparison over Infiniband is presented in Figure 3. The maximum bandwidth achieved by MPJ Express running over *ibdev* is 16 Gbps for 2 Mega Bytes message size. Whereas SDP achieve peak bandwidth of 7.4 Gbps and IPoIB has nominal 1.1 Gbps peak bandwidth for 2 Mega Byte message size. *ibdev* shows significant improvement over SDP and IPoIB as *ibdev* is low level device which makes full use of IB network. The poor performance of SDP and IPoIB is due to the extra overhead incurred by Upper Layer Protocols.

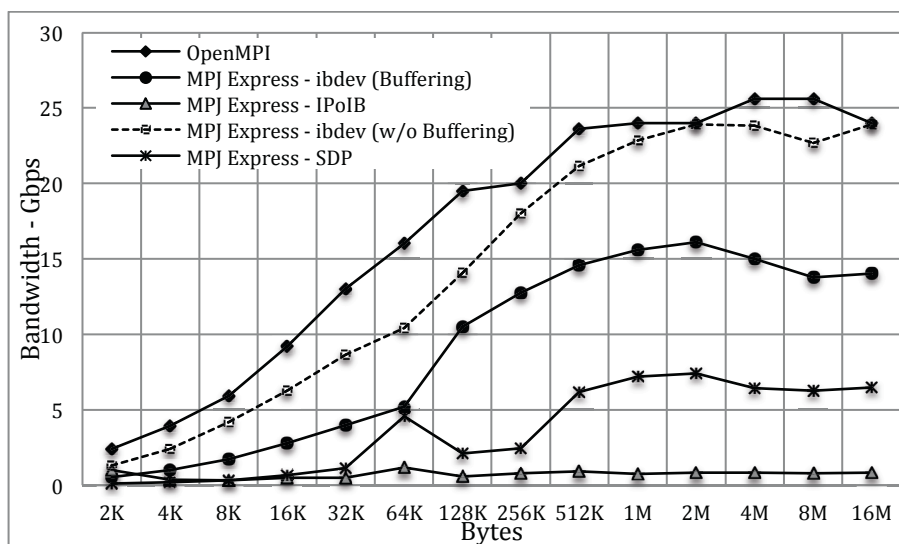


Figure 3: Bandwidth of Message Passing Libraries

OpenMPI achieved 26 Gbps throughput for 8 Mega Bytes message size which is better than any of the Java based devices. It can also be seen that throughput of MPJ Express devices started to drop for message size greater than 2 Mega Bytes. This is due to the internal buffer layer of MPJ Express as it copies message into internal buffer. The result of MPJ Express *ibdev* without buffering layer shows throughput performance closer to the OpenMPI.

5.2 NAS Parallel Benchmarks (NPB)

NAS Parallel Benchmarks are a set of benchmarks designed for performance evaluation of highly parallel computing environments. These benchmarks are derived primarily from Computational Fluid Dynamics (CFD) applications. They evaluate different patterns of computation and communication across a wide range of applications.

The impact of *ibdev* on the performance of NPB implementations of MPJ Express has also been evaluated. Three problems have been selected: CG (Conjugate Gradient), FT (Fourier Transform), and IS (Integer Sort). Each of these problems represents very different communication and memory access patterns. CG is a sparse iterative solver and exhibits irregular memory access and communication. FT performs a series of 3-D FFTs and tests all-to-all communication. IS does random memory access and aggregated communication. Figure 4, 5 and 6 shows the NPB results for a workload of class C. The results are presented in terms of MOPS (Millions of Operations per Seconds). Test runs have been conducted with MPJ Express running *ibdev*, *niodev* on SDP, and *niodev* on IPoIB.

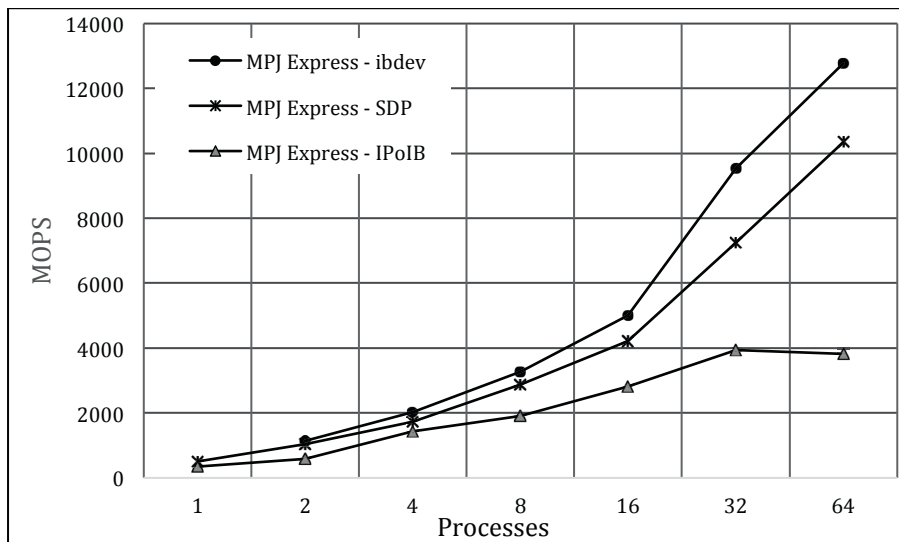


Figure 4: CG (Class C) result over InfiniBand

For CG, *ibdev* has a much better performance compared to *niodev* running on either emulation layer. Also for FT benchmark, *ibdev* shows an almost 2.5 times increase in performance compared to *niodev* (running on IPoIB) and 1.5 times *niodev* (running on SDP). For IS, the results of *niodev* with IPoIB show a substantial drop in performance when running on 64 processes.

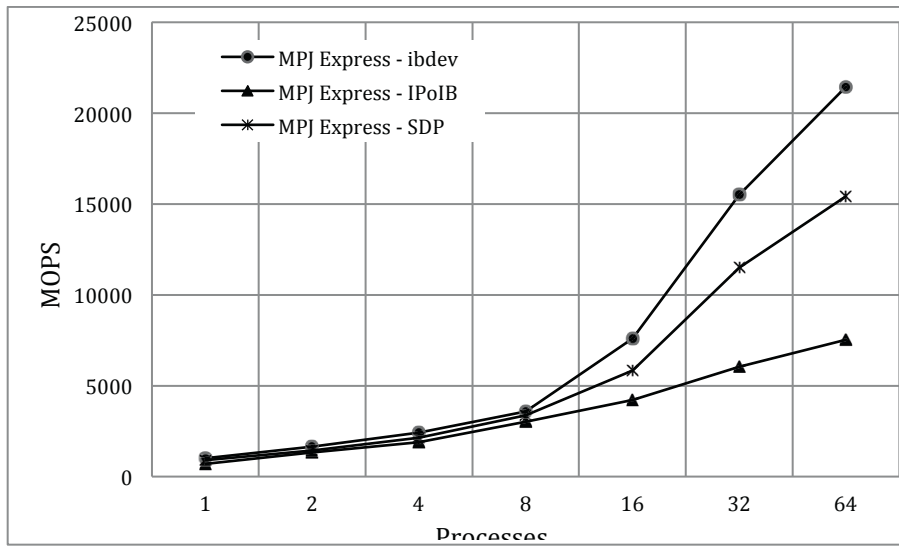


Figure 5: FT (Class C) result over InfiniBand

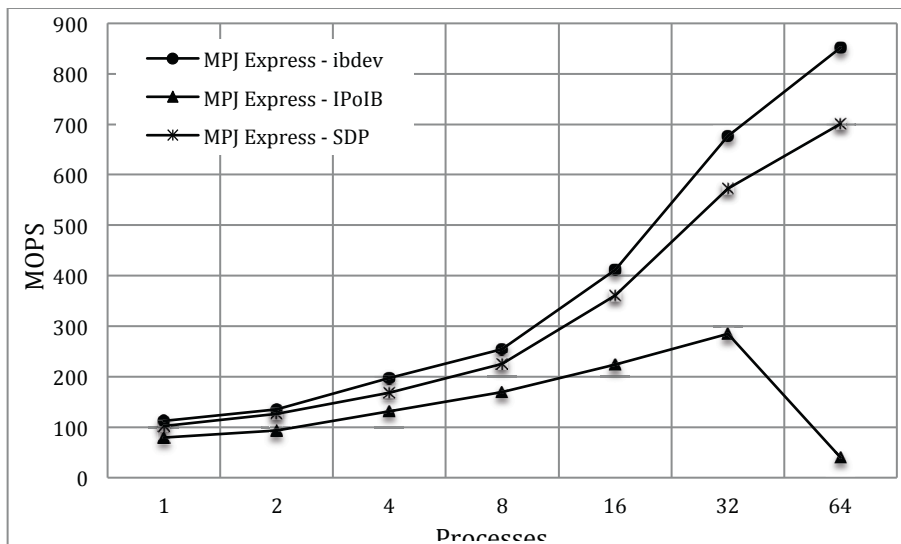


Figure 6: IS (Class C) result over InfiniBand

6 Conclusions and Future work

This paper presented the design of *ibdev*, a low-level communication device for InfiniBand networks. It provides Java message passing applications with a high performance because of its direct reliance on RDMA support in InfiniBand.

The performance evaluations show that *ibdev* achieves much better start-up latencies and bandwidth compared to *niodev*. Our performance evaluation reveals that MPJ Express achieves latency of 27 micro-seconds using *ibdev* compared to 51 micro-seconds using IPoIB—for one byte

message. On the other hand, MPJ Express achieved throughput of 16 Gbps using *ibdev* as compared to 7.4 Gbps using SDP—for 2 Mega Byte message. The impact of *ibdev* when running NPB's is also significant. The NPB's results also highlight the fact that MPJ Express can achieve better scalability and speedup, when efficient low level communication libraries are made available. We have not evaluated the impact of *ibdev* on the processor utilization, but it can be a very useful benchmark.

In future we plan to add support for shared memory for processes running on the same machine. Currently, it is a pure networking device and uses network communication for processes sharing memory. One worthy future enhancement can be incorporation of shared memory in *ibdev*. This would further increase the scalability and performance of the device.

References

1. Guillermo L. Taboada, Sabela Ramos, Roberto R. Expósito, Juan Touriño, and Ramón Doallo. 2013. Java in the High Performance Computing arena: Research, practice and experience. *Sci. Comput. Program.* 78, 5 (May 2013), 425-444.
2. Brian Blount and Siddhartha Chatterjee. 1999. An evaluation of Java for numerical computing. *Sci. Program.* 7, 2 (April 1999), 97-110.
3. Roberto R. Expósito, Guillermo L. Taboada, Juan Touriño, and Ramón Doallo. 2012. Design of scalable Java message-passing communications over InfiniBand. *J. Supercomput.* 61, 1 (July 2012), 141-165.
4. Guillermo L. Taboada, Juan Tourino, Ram Doallo, Yao Lin, and Jizhong Han. 2009. Efficient Java Communication Libraries over InfiniBand. In *Proceedings of the 2009 11th IEEE International Conference on High Performance Computing and Communications (HPCC '09)*. IEEE Computer Society, Washington, DC, USA, 329-338.
5. Guillermo L. Taboada, Juan Touriño, and Ramón Doallo. 2012. F-MPJ: scalable Java message-passing communications on parallel systems. *J. Supercomput.* 60, 1 (April 2012), 117-140.
6. Guillermo L. Taboada, Sabela Ramos, Roberto R. Expósito, Juan Touriño, and Ramón Doallo. 2013. Java in the High Performance Computing arena: Research, practice and experience. *Sci. Comput. Program.* 78, 5 (May 2013), 425-444.
7. Hongwei Zhang, Wan Huang, Jizhong Han, Jin He, and Lisheng Zhang. 2007. A Performance Study of Java Communication Stacks over InfiniBand and Giga-bit Ethernet. In *Proceedings of the 2007 IFIP International Conference on Network and Parallel Computing Workshops (NPC '07)*. IEEE Computer Society, Washington, DC, USA, 602-607.
8. Intel. 2013. Optimizing Java and Apache Hadoop for Intel Architecture. White Paper. URL: <http://hadoop.intel.com/pdfs/big-data-optimizing-java-apache-hadoop-for-intel-architecture-white-paper.pdf>
9. Weihang Jiang, Jiuxing Liu, Hyun-Wook Jin, D. K. Panda, W. Gropp, and R. Thakur. 2004. High performance MPI-2 one-sided communication over InfiniBand. In *Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid (CCGRID '04)*. IEEE Computer Society, Washington, DC, USA, 531-538.
10. Jiuxing Liu, Jiesheng Wu, Sushmitha P. Kini, Pete Wyckoff, and Dhabaleswar K. Panda. 2003. High performance RDMA-based MPI implementation over InfiniBand. In *Proceedings of the 17th annual international conference on Supercomputing (ICS '03)*. ACM, New York, NY, USA, 295-304.
11. Mark Baker, Bryan Carpenter, Aamir Shafi. 2006. MPJ Express: Towards Thread Safe Java HPC. *IEEE International Conference on Cluster Computing*, Sept. 2006. 1-10.

12. Markus Bornemann, Rob V. van Nieuwpoort, and Thilo Kielmann. 2005. MPJ/Ibis: a flexible and efficient message passing platform for java. In *Proceedings of the 12th European PVM/MPI users' group conference* (PVM/MPI'05), Berlin, Heidelberg, 217-224.
13. Bryan Carpenter, Glenn Judd, Anthony Skjellum, and Geoffrey Fox. 2000. MPJ: MPI-like Message Passing for Java. *Concurrency and Computation: Practice and Experience*, vol. 12, no. 11, 1019–1038.
14. Wan Huang, Hongwei Zhang, Jin He, Jizhong Han, and Lisheng Zhang. 2007. Jdib: Java Applications Interface to Unshackle the Communication Capabilities of InfiniBand Networks. In *Proceedings of the 2007 IFIP International Conference on Network and Parallel Computing Workshops* (NPC '07). IEEE Computer Society, Washington, DC, USA, 596-601.
15. Wan Huang, Jizhong Han, Jin He, Lisheng Zhang, and Yao Lin. 2008. Enabling RDMA Capability of InfiniBand Network for Java Applications. In *Proceedings of the 2008 International Conference on Networking, Architecture, and Storage* (NAS '08). IEEE Computer Society, Washington, DC, USA, 187-188.
16. Bryan Carpenter, Geoffrey Fox, Sung-Hoon Ko, Sang Lim. 2013. Specification, mpiJava 1.2: API. URL: <http://www.hpjava.org/reports/mpiJava-spec/mpiJava-spec/mpiJava-spec.html>.